ارائه یک روش آزمون مبتنی‌برمدل در سطح معماری نرم‌افزار با استفاده از آتاماتای تیمی

سمانه واعظ دلیلی[1]، سید مهران شرفی[2]

[1] دانشکده مهندسی کامپیوتر، دانشگاه آزاد اسلامی واحد نجف‌آباد، s_vaez@sco.iaun.ac.ir

[2] دانشکده مهندسی کامپیوتر، دانشگاه آزاد اسلامی واحد نجف‌آباد، Mehran_sharafi@iaun.ac.ir

چکیده

در یک سیستم مبتنی بر مؤلفه، صحّت سیستم وابسته به درستی تک تک مؤلفه‌ها و تعاملات بین آنهاست. در این سیستم‌ها اگر فرض کنیم، هر کدام از مؤلفه‌ها به صورت صحیح کار می‌کنند، بعضی از خطاها ممکن است، هنگامی به وجود آیند که مؤلفه‌ها یکپارچه می‌شوند. آزمون مبتنی بر مدل، روشی نوین است، که با ایجاد خودکار نمونه‌های آزمون از مدل رفتاری سیستم، خطاها را کشف می‌کند. در این مقاله، با استفاده از آزمون مبتنی بر مدل، صحّت تعاملات و ارتباطات مؤلفه‌ها مورد ارزیابی قرار می‌گیرد. در روش پیشنهادی، از توصیفات نیمه‌رسمی معماری نرم‌افزار به عنوان مدل رفتاری سیستم استفاده می‌شود؛ برای ایجاد خودکار نمونه-های آزمون، توصیفات معماری نرم‌افزار با آتاماتای تیمی رسمی می‌شوند. از توصیفات رسمی معماری نرم‌افزار مجموعه‌ای از نمونه-های آزمون استخراج می‌شوند. این نمونه‌های آزمون، قابلیت اجرا در سیستم را ندارند، بنابراین به نمونه‌های آزمون واقعی تبدیل می‌گردند. در نهایت هم الگوریتمی برای آزمون مرکب در سیستم‌های مبتنی بر مؤلفه ارائه می‌شود. اعمال راهکار پیشنهادی روی یک سیستم منابع انسانی نمونه «منابع انسانی امین»، نشان می‌دهند که روش پیشنهادی نقش بسزایی در بهبود چالش‌های آزمون نرم‌افزار ایفا می‌کند.

کلمات کلیدی

آزمون نرم‌افزار، آزمون مبتنی بر مدل، معماری نرم‌افزار، آتاماتای تیمی

۱- مقدمه

آزمون نرم‌افزار[1]، رفتارهای اجرایی برنامه را که توسط مجموعه‌ای از نمونه‌های آزمون[2] انتخاب شده‌اند، در برابر رفتارهای مورد انتظار، ارزیابی می‌کند[2]. هدف از آزمون نرم‌افزار، کشف خطاهاست.

آزمون نرم‌افزار نقش بسیار مهمی در کیفیت نرم‌افزار ایفا می‌کند. با پیچیده‌ترشدن سیستم‌های نرم‌افزاری، فرآیند آزمون نرم‌افزار نیز پیچیده‌تر شده و به تدریج به عنوان یک فرآیند پرهزینه و زمان بر، ظهور یافته‌است. به همین دلیل است که راه حل‌های زیادی برای خودکارسازی فرآیند آزمون نرم افزار مطرح شده‌است.

با ظهور توصیفات نرم‌افزار، کُد، تنها منبع برای آزمون نرم‌افزار نیست، بلکه توصیفات نرم‌افزار نیز می‌توانند برای این منظور استفاده‌شوند.

معماری نرم‌افزار[3] یک محصول نرم‌افزاری را براساس مؤلفه (عناصر محاسباتی)، اتصالات (عناصر ارتباط دهنده) و پیکربندی (سازمان کلی) توصیف می کند[3].

در سال‌های اخیر، استفاده از معماری در سطوح مختلف آزمون نرم-افزار افزایش یافته‌است، زیرا معماری نرم‌افزار به عنوان اولین محصول طراحی بهترین تجرید از سیستم را توصیف می‌کند.

در مرجع[10] محققان روشی برای آزمون جریان داده و جریان کنترل در سطح معماری نرم‌افزار پیشنهاد کرده‌اند. این محققان روابطی بین عناصر معماری تعریف کرده و از آنها برای آزمایش مسیرهای معماری استفاده نموده‌اند. در مرجع [6] نویسندگان از توصیفات معماری نرم‌افزار، در آزمون رگرسیون استفاده کرده‌اند. این نویسندگان از معماری نرم‌افزار برای کاهش هزینه آزمون نرم‌افزار در هنگام تغییر سیستم بهره می‌برند.

در مرجع [7] نویسندگان از معماری نرم‌افزار در آزمون سطح تجمیع برای سیستم‌های همروند بهره برده‌اند. در مرجع [8] با استفاده از سیستم گذارهای برچسب دار[4]، معماری نرم‌افزار توصیف می‌شود و سپس نمونه‌های آزمون انتزاعی از توصیفات رسمی معماری نرم‌افزار استخراج می‌گردد، در نهایت ، نمونه‌های آزمون انتزاعی برای اجرای آزمایش به نمونه های آزمون واقعی تبدیل می‌شوند.

یک سیستم مبتنی بر مؤلفه، از کنار هم قرارگرفتن مؤلفه‌ها به وجود می‌آید و گاهی اوقات، سازندگان حتی از جزئیات مؤلفه‌ها نیز بی‌اطلاعند. اگر فرض کنیم که هر مؤلفه بصورت صحیح کار می‌کند، برای صحّت سیستم، هر مؤلفه باید بصورت صحیح سرویس‌های مورد نیاز را از مؤلفه‌های دیگر درخواست کرده و سرویس‌ها را بصورت صحیح برای مؤلفه‌های دیگر فراهم نماید. به دلیل آن‌که در یک سیستم مبتنی بر مؤلفه، ارتباطات و تعاملات گسترده‌ای وجود دارد، آزمون نرم‌افزار، با چالش‌های زیادی روبرو خواهد بود.

امروزه، مدلسازی، نقش حیاتی در فرآیند ایجاد نرم‌افزار بر عهده دارد. به دلیل اهمیت بهره‌برداری از مدل‌ها و سودمندی آنها، روشهای گوناگونی پیشنهاد شده‌اند که از مدل‌ها در آزمون نرم‌افزار استفاده می‌نمایند. به این روش‌ها در ادبیات آزمون نرم‌افزار، آزمون مبتنی بر مدل[۵] گفته می‌شود [4]. آزمون مبتنی بر مدل، روشی است که مقابله با تمامی چالش‌های آزمون نرم‌افزار را نوید می‌دهد. آزمون مبتنی بر مدل با استفاده از یک مدل مرجع، صحّت سیستم را مورد ارزیابی قرارمی‌دهد. آزمون مبتنی بر مدل، از طریق توصیفات رسمی مدل، قادر به خودکارسازی فرآیند آزمون نرم‌افزار خواهدبود. هر چه مدل مرجع سیستم را بهتر توصیف کند، آزمون مبتنی بر مدل بهتر ایفای نقش خواهدکرد.

مهمترین، مزیت آزمون مبتنی بر مدل در برابر سایر مدل‌ها ایجاد خودکار نمونه‌های آزمون است و مهمترین نقص آزمون مبتنی بر مدل این است که ساختار و توصیفات مدل مرجع باید به صورت رسمی موجود باشد [4].

محققان در [11] روشی نوین برای ایجاد نمونه‌های آزمون ارائه کرده‌اند. آنها از توصیفات رسمی معماری نرم‌افزار در آزمون مبتنی بر مدل، استفاده نموده‌اند. در این روش پیشنهادی، توصیفات معماری نرم‌افزار در ابتدا به صورت acme [12] هستند و برای ایجاد نمونه‌های آزمون، توصیفات معماری نرم‌افزار با شبکه‌های پتری رسمی می‌شود.

در این مقاله، از معماری نرم‌افزار به عنوان مدل مرجع در آزمون مبتنی بر مدل استفاده می‌شود که ایده‌ی اصلی آن از مرجع[11] الهام گرفته‌شده‌است.

با وجود آن‌که توصیفات رسمی، قابلیت خودکارسازی در آزمون نرم‌افزار را فراهم می‌کنند، اما اکثر افراد تعاریف ساده و نیمه‌رسمی را ترجیح می‌دهند. فرم رسمی مدل‌ها بندرت در فرآیند ایجاد نرم‌افزار مور استفاده قرار می‌گیرد و معمولاً فرم نیمه‌رسمی مدل‌ها در چرخه‌ی حیات نرم‌افزار ایجاد می‌شود.

UML[۶] [15] یکی از زبان‌های استاندارد مدلسازی است که توسط دامنه‌ی وسیعی از سازندگان نرم‌افزار مورد استفاده قرار می‌گیرد، نمودارهای UML از قابلیت درک بالایی برخوردارند، همچنین نگارش‌های جدید UML آن را برای توصیفات معماری نرم‌افزار مناسب‌تر ساخته‌است.

به جهت کاربردی بودن روش پیشنهادی، از توصیفات نیمه‌رسمی معماری نرم‌افزار بهره‌مند می‌شویم که به کمک نمودارهای UML2.0 ارائه شده‌اند، و فرض بر آن است که این توصیفات در چرخه‌ی حیات نرم‌افزار ایجاد شده‌اند و در روش پیشنهادی صرفاً از آنها استفاده می‌شود.

آتاماتای تیمی[۷][9] یک مدل مبتنی بر آتاماتا است، که به جهت ویژگی‌های منحصر به فردی که دارد، برای توصیف معماری نرم‌افزار مناسب است[13]. در روش پیشنهادی آتاماتای تیمی منحصر به فرد، برای رسمی‌سازی معماری نرم‌افزار تعریف می‌شود.

بعد از رسمی‌سازی معماری نرم‌افزار، نمونه‌های آزمون از معماری نرم‌افزار استخراج می‌گردند؛ نمونه‌های آزمون حاصل از توصیفات معماری نمونه‌های آزمون انتزاعی[۸] نامیده می‌شوند. از طریق اصلاح و پالایش نمونه‌های آزمون انتزاعی، نمونه‌های آزمون واقعی ایجاد می‌شود. نمونه‌های آزمون واقعی را می‌توان روی سیستم اجرا کرد و نتایج اجرا را مشاهده و تحلیل کرد.

ساختار این مقاله بدین شرح است: در بخش دوم روش پیشنهادی شرح داده می‌شود. در بخش سوم، یک سیستم منابع انسانی نمونه با استفاده از روش پیشنهادی آزمایش می‌شود. الگوریتمی برای آزمون مرکب در سیستم‌های مبتنی بر مؤلفه در بخش چهارم ارائه می‌شود. نتیجه‌گیری و کارهای آینده هم دربخش چهارم ذکر می‌شود. در پایان هم مراجع استفاده شده در این مقاله معرفی می‌شوند.

۲- روش پیشنهادی

در روش پیشنهادی فرض بر آن است که معماری نرم‌افزار به صورت UML2.0 درسیستم موجود است و نمودارهای ترتیب[۹] و مؤلفه‌ی[۱۰] UML2.0 به ترتیب جنبه‌های رفتاری و ساختاری معماری نرم‌افزار را توصیف می‌کنند. همچنین رفتار داخلی مؤلفه‌ها بصورت نمودارهای ماشین وضعیت[۱۱] در مستندات سیستم موجود است و آتاماتای تیمی SA-TA برای مدلسازی اتصالات اسمبلی[۱۲][14] ارائه می‌شود. برای اتصالات وکالتی[۱۳][14] بین مؤلفه‌ها الگوریتم رسمی‌سازی ترکیبی در بخش چهارم ارائه خواهد شد. مراحل روش پیشنهادی بدین شرح است:

۲-۱- مرحله اول: استخراج توصیفات معماری نرم‌افزار با توجه با سناریوهای کاربر

در هر سیستم نرم‌افزاری آزمون کامل امکان پذیر نیست[5] و آزمون‌گیرنده، قادر به ارزیابی بخشی از سیستم خواهد بود. در روش پیشنهادی قسمت‌هایی از سیستم که از نظر کاربر مهم‌تر هستند، ارزیابی می‌شوند. در روش پیشنهادی، با استفاده از سناریوهای کاربر قسمت‌های مهم‌تر معماری نرم‌افزار استخراج و رسمی می‌شوند. سناریوهای کاربر، با زبان طبیعی و معمولاً غیررسمی بیان می‌شود. تفسیر سناریوی کاربر و استخراج جنبه‌های ساختاری و رفتاری معماری نرم‌افزار متناسب با سناریوی کاربر بر عهده‌ی معمار نرم‌افزار خواهد بود. معمار نرم‌افزار، برای هر سناریوی کاربر نمودار ترتیب و

نمودار مؤلفه‌ی UML2.0 را از توصیفات معماری نرم‌افزار استخراج می‌کند.

در روش پیشنهادی برای رسمی کردن توصیفات معماری نرم‌افزار هر نمودار مؤلفه UML2.0 به صورت (UcomponentsCD,Uassemblyconnector) تعریف می‌شود. که UcomponentsCD مجموعه‌ی متناهی از مؤلفه‌ها در نمودار مؤلفه و Uassemblyconnector نماد اتصالات اسمبلی بین مؤلفه‌هاست. اتصالات اسمبلی نیز به صورت (C_r, C_p, I) تعریف می‌شود که C_p مؤلفه‌ی فراهم‌کننده[۱۴] و C_r مؤلفه‌ی درخواست کننده[۱۵] است.

هر نمودار ترتیب UML2.0 به صورت (UcomponentsSD,Stimuli) تعریف می‌شود که UcomponentsSD مجموعه‌ی متناهی از مؤلفه‌ها در نمودار ترتیب است و Stimuli به صورت (Uassemblyconnector,Msg) خواهد بود، Uassemblyconnector همان اتصالات اسمبلی است و Msg پیام بین مؤلفه‌ها در نمودار ترتیب است. این توصیفات در ابتدا در[۱] مطرح شده است و ما برای رسمی‌سازی معماری نرم‌افزار آنها را بهبود داده‌ایم.

۲-۲- مرحله دوم: رسمی‌سازی معماری نرم‌افزار توسط آتاماتای تیمی

آتاماتای تیمی از مجموعه‌ای از مؤلفه‌های آتاماتا[۱۶] تشکیل شده‌است و مؤلفه‌های آتاماتا با یکدیگر تعامل برقرار می‌کنند و آتاماتای تیمی را تشکیل می‌دهند. در تعاملات بین مؤلفه‌ها، شرکت حداقل یک مؤلفه آتاماتا در همزمانی کنش مشترک کافی است و همزمانی‌های بین مؤلفه‌ها به صورت خاص تعریف نمی‌شود، بلکه در همزمانی کنش مشترک مجموعه‌ای از آتاماتای تیمی تعریف می‌شود که در همزمانی کنش‌های مشترک متفاوت هستند. این ویژگی، قابلیت توصیف هر نوع تعاملی را بین مؤلفه های آتاماتا فراهم می‌کند و آتاماتای تیمی را برای توصیف معماری نرم‌افزار مناسب می‌سازد. بنابراین برای ایجاد یک آتاماتای تیمی منحصر به فرد همزمانی‌های خاصی باید انتخاب شود. تعریف آتاماتای تیمی در حوزه‌ی این مقاله نمی‌گنجد و خوانندگان برای درک بیشتر می‌توانند به [9] مراجعه کنند.

در مرجع [۱] الگوریتم UML2TA برای تبدیل توصیفات نیمه‌رسمی معماری نرم‌افزار به آتاماتای تیمی ارائه شده‌است. امّا دراین الگوریتم وضعیت هر مؤلفه آتاماتا فقط با یک سناریوی کاربری در نظر گرفته می‌شود، در حالی که هر مؤلفه در هنگام آزمون نرم‌افزار ممکن است به مؤلفه‌های دیگری غیر از مؤلفه‌های سناریو سرویس بدهد یا سرویس درخواست کند، بنابراین در روش پیشنهادی رفتارهای داخلی هر مؤلفه به صورت نمودارهای ماشین وضعیت، در نظر گرفته‌می‌شود که وضعیت‌ها در ماشین وضعیت حالت‌های بالقوه‌ای هستند که مؤلفه در هنگام درخواست‌کردن و یا فراهم‌کردن سرویس در این وضعیت‌ها قرار می‌گیرد.

در این مرحله، هر ماشین وضعیت به مؤلفه آتاماتا مدل می‌شود و آتاماتای تیمی منحصر به فرد SA-TA از همزمانی بین مؤلفه‌های آتاماتا با توجه به تعاملات مؤلفه‌ها در معماری نرم‌افزار بدست می‌آید. البته این نکته قابل ذکر است که ایجاد آتاماتای تیمی برای هر سناریو بصورت جداگانه انجام خواهد شد. در این‌جا لازم است بعضی از نمادهایی که در رسمی‌سازی معماری نرم‌افزار، استفاده شده‌است معرفی شوند.

۲-۲-۱- نمادها

فرض کنید که N مجموعه‌ی اعداد طبیعی مثبت را نشان می‌دهد و مجموعه‌ی $\{1,2,...,l\}$، مجموعه‌ی اندیس‌ها را نمایش می‌دهد که $l \subseteq N$ است. آنگاه ضرب کارتزین بصورت $\prod_{i\in l} V_i$ نشان داده می‌شود و $proj_j : \prod_{i\in l} V_i \to V_j$ بصورت تابعی از V_i به روی V_j است، به طوریکه $proj_j((a_1,.....,a_n)) = a_j$ است.

۲-۲-۲- توصیف هر مؤلفه سیستم به صورت مؤلفه آتاماتا

در روش پیشنهادی، هر ماشین وضعیت بصورت یک مؤلفه‌ی آتاماتا مدل می‌شود. هر مؤلفه‌ی آتاماتا بین کنش‌های ورودی، خروجی و داخلی تمایز قائل می شود[9]. بنابراین برای مدل کردن هر ماشین وضعیت به صورت مؤلفه‌آتاماتا لازم است، کنش‌های ماشین وضعیت به سه زیرمجموعه‌ی ورودی، خروجی و داخلی تقسیم شود. این تقسیم بندی بر اساس نمودار ترتیب و نمودار مؤلفه، در تعاریف (۱)، (۲) و (۳) نمایش داده‌شده‌است. در این تعاریف $\sum_j$ مجموعه‌ی همه‌ی کنش‌ها در ماشین وضعیت j را نمایش می‌دهد و a هم نماینده یکی از کنش‌ها در ماشین وضعیت j است. $\sum_{j,inp}$ نماد مجموعه‌ی کنش‌های ورودی مؤلفه‌ی آتاماتای j و $\sum_{j,out}$ مجموعه‌ی کنش‌های خروجی مؤلفه‌ی آتاماتای j را نمایش می‌دهد. در صورتی که کنشی، کنش ورودی یا خروجی نباشد آن کنش، کنش داخلی درنظر گرفته می‌شود و با $\sum_{j,\text{int}}$ نمایش داده می‌شود.

$$\sum_{j,inp} = \{a \in \sum_j \mid \exists s \in stimuli: a \in s.Msg, \exists m \in \text{Uassemblyconnector}: j \in m.C_p, j \in U\text{componentsSD}, j \in U\text{componentsCD}\}$$

(۱) کنش ورودی

$$\sum_{j,out} = \{a \in \sum_j \mid \exists s \in stimuli: a \in s.Msg, \exists m \in \text{Uassemblyconnector}: j \in m.C_r, j \in U\text{componentsSD}, j \in U\text{componentsCD}\}$$

(۲) کنش خروجی

$$\sum_{j,\text{int}} = \{a \in \sum_j \setminus (\sum_{j,out} \cup \sum_{j,inp})\}$$

(۳) کنش داخلی

در آتاماتای تیمی کنش‌های داخلی هر مؤلفه آتاماتا منحصر به فرد است[9]. بنابراین اگر کنش داخلی در یک مؤلفه آتاماتا در کنش‌های داخلی مؤلفه‌های دیگر تکرار شد، آن‌گاه آن کنش داخلی تغییر نام پیدا خواهد کرد. مجموعه‌ی کنش‌های تکراری برای هر مؤلفه بصورت تعریف (۴) بیان می‌شوند.

$$R_j = \{a \in \sum_{j,\text{int}} \mid \exists i, j \in l : \exists a \in (\sum_{i,\text{int}} \cap \sum_{j,\text{int}})\}$$

(۴)مجموعه‌ی کنش‌های تکراری

۲-۲-۳- ایجاد آتاماتای تیمی

در این مرحله، آتاماتای تیمی SA-TA برای رسمی‌سازی توصیفات معماری نرم‌افزار تعریف می‌شود. که این آتاماتا درتعریف(۶) نمایش داده‌شده‌است. در آتاماتای تیمی کنش‌های داخلی در تعاملات بین مؤلفه‌ها شرکت نمی‌کنند و کنش‌های خارجی(مجموعه‌ای از کنش‌های ورودی و خروجی) هستند که تعاملات بین مؤلفه‌های آتاماتا و پیرامونش را برقرار می‌کنند. برای ایجاد آتاماتای تیمی SA-TA کنش‌های خارجی به طریقی انتخاب می‌شوند که کنش مشترک برای مؤلفه فراهم کننده، نقش ورودی و برای مؤلفه درخواست‌کننده، نقش خروجی، داشته‌باشد[۱۷] . کنش‌های خارجی در تعریف (۶) با $\sum_{ext}$ نمایش داده شده‌است و کنش‌های داخلی برابر فضای کامل گذار[۱۸] [9] ($\Delta_a(\delta)$) تعریف می‌شود و در تعریف (۶) با $\sum_{\text{int}}$ نمایش داده شده‌است. فضای کامل گذار مجموعه‌ای از تعاملات بین مؤلفه‌های آتاماتا است که حداقل یک مؤلفه‌آتاماتا در تعاملات کنش مشترک شرکت می‌کند. $l_{a,inp}$ دامنه‌ای از مؤلفه‌ها است که کنش a نقش ورودی را برای آنها ایفا می‌کند و $l_{a,out}$ دامنه‌ای از مؤلفه‌ها است که a نقش خروجی را برای آنها دارد[9]. در SA-TA، کنش های ارتباطی بین مؤلفه‌ها [9] ($\sum_{com}$) به عنوان کنش داخلی در نظر-گرفته‌می‌شود، بنابراین، کنش‌های ارتباطی برای بقیه‌ی مؤلفه‌های آتاماتاها غیر قابل مشاهده خواهند بود. از این ایده، بعداً برای استفاده از آتاماتای تیمی به عنوان مؤلفهٔ آتاماتا در سطوح بالاتر در الگوریتم رسمی‌سازی ترکیبی استفاده خواهیم کرد.

$$\tau^{SA} = (\prod_{i\in l} Q_i, (\sum_{inp}, \sum_{out}, \sum_{\text{int}}), \delta, \prod_{i\in l} I_i)$$

$$\sum_{\text{int}} = (\bigcup_{i\in l} \sum_{i,\text{int}}) \cup \sum_{com},$$

$$\sum_{out} = (\bigcup_{i\in l} \sum_{i,out}) \setminus \sum_{com},$$

$$\sum_{inp} = (\bigcup_{i\in l} \sum_{i,inp}) \setminus \sum_{com}$$

$$\text{if } a \in \sum_{\text{int}}, \delta_a = \Delta_a(\delta)$$

$$\text{if } a \in \sum_{ext}$$

$$\delta_a = \{(q,q') \in \Delta_a(\delta) \mid proj^{[2]}_{l_{a,out}}(q,q') \in \Delta_a(\{C_i \mid i \in l_{a,out}\}), proj^{[2]}_{l_{a,inp}}(q,q') \in \Delta_a(\{C_i \mid i \in l_{a,inp}\})\}$$

(۶)آتاماتای تیمی SA-TA

۲-۳- مرحلهٔ سوم: استخراج نمونه های آزمون معماری نرم‌افزار

در این مرحله نمونه‌های آزمون از توصیفات رسمی معماری نرم-افزار(آتاماتای تیمی) استخراج می‌شوند. این نمونه‌های آزمون، انتزاعی هستند و قابلیت اجرا روی سیستم را ندارند و نیاز به اصلاح و پالایش دارند که در مرحلهٔ بعدی انجام خواهد شد. هر نمونهٔ انتزاعی در واقع مسیری روی SA-TA است.

مسیرهای معماری که از آتاماتای تیمی استخراج می شود، گاه بی-نهایت هستند. در روش پیشنهادی، از معیارشمول[۱۹] برای جلوگیری از تولید بی‌نهایت نمونه‌های آزمون استفاده شده است و سه معیار شمول معرفی می‌شود.

معیار وضعیت: هر وضعیتی در SA-TA حداقل یکبار پیمایش شود.

معیار گذار: همهٔ گذارها حداقل یکبار پیمایش شوند.

معیار مسیر: همهٔ مسیرها حداقل یکبار پیمایش شوند.

۲-۴- مرحله چهارم: ایجاد نمونه های آزمون واقعی

برای ایجاد نمونه‌های آزمون واقعی، نگاشتی بین مؤلفه ها و کنش-های معماری نرم‌افزار و سطوح پایین پیاده سازی برقرار می‌گردد. دراین مرحله، آزمون گیرنده با درکی که از پیاده سازی دارد. نمونه‌های آزمون انتزاعی را به توابع و ساختارهای پیاده سازی تبدیل می‌کند.

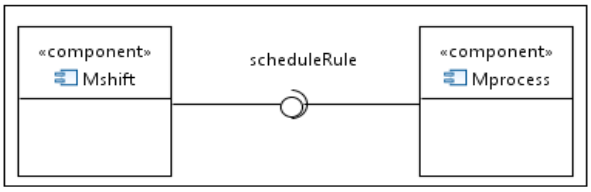
۳- نمونه مطالعاتی

سیستم جامع منابع انسانی امین، سیستمی است که برای بکارگیری روش پیشنهادی در این مقاله انتخاب شده‌است. هدف این سیستم

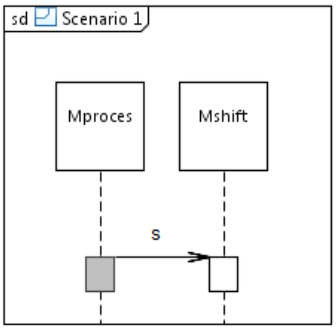
خودکارسازی تمامی فرآیندهای درون سازمانی است و همچنین مدیریت منابع انسانی از طریق ثبت داده‌های کارکنان و پردازش و تفسیر فیلدهای اطلاعاتی می‌باشد. سیستم منابع انسانی امین، از مجموعه‌ای از مؤلفه‌ها تشکیل شده‌است که هر کدام از مؤلفه‌ها وظیفه‌ی خاصی را برعهده دارند. همچنین بعضی از مؤلفه‌ها برای انجام وظیفه‌ی خود از زیر مؤلفه‌هایی تشکیل شده‌اند(مؤلفه مرکب[۲۰]). این سیستم از طریق ارتباطات و تعاملات مؤلفه‌ها سرویس‌های لازم برای کاربران را فراهم می‌کند. همچنین توصیفات معماری نرم‌افزار به صورت UML2.0 در سیستم موجود است. توصیفات معماری نرم‌افزار نحوه‌ی ارتباطات مؤلفه‌ها و زیر سیستم‌های مختلف و همچنین ساختار کلی این سیستم را توصیف می‌کنند. بحث در مورد معماری این سیستم در حوزه‌ی این مقاله نمی‌گنجد.

فرض کنید که کاربر (این کاربر می تواند مدیر باشد) می‌خواهد، متابعت کارکنان از قوانین تردد سازمان را با استفاده از سیستم جامع منابع انسانی امین بررسی کند. این سناریو را شماره ۱ می‌نامیم.

با توجه به تفاسیر معمار نرم‌افزار از سناریوی شماره ۱ نمودار مؤلفه مطابق شکل (۱) و نمودار ترتیب مطابق شکل (۲) خواهد بود.

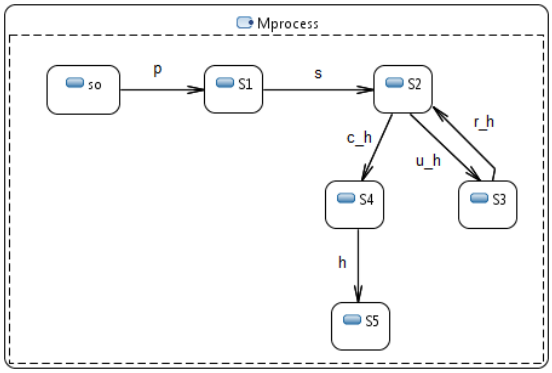


شکل (۱): نمودار مؤلفه سناریوی شماره۱

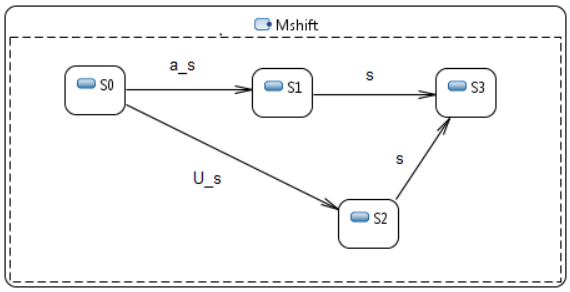


شکل (۲): نمودار ترتیب سناریوی شماره ۲

ماشین‌های وضعیت مؤلفه‌های موجود در سناریوی شماره ۱ در شکل‌های (۳)و(۴) نشان داده شده است.



شکل (۳): ماشین وضعیت مؤلفهMprocess



شکل (۴): ماشین وضعیت مؤلفه‌ی Mshift

برای ایجاد مؤلفه‌ی آتاماتای Mprocess و Mshift کنش‌های ماشین وضعیت شکل (۳) و (۴) را به سه زیرمجموعه تقسیم می‌کنیم.

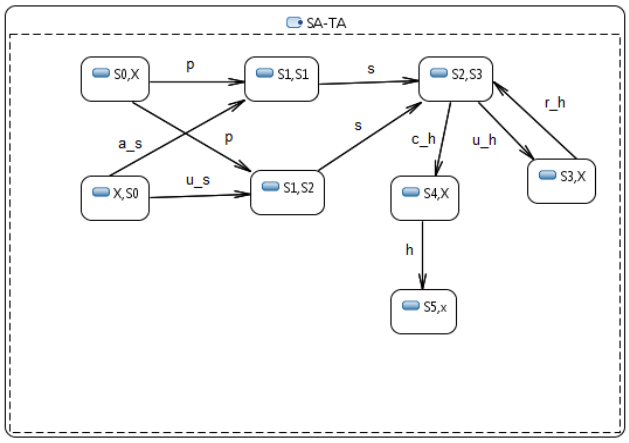
کنش s در نمودار ترتیب در شکل(۲) موجود است و همچنین مؤلفه‌ی Mprocess و Mshift از جمله‌ی مؤلفه‌های سناریوی شماره ۱ محسوب می‌شود و در نمودار مؤلفه و ترتیب موجود است. همچنین مؤلفه‌یMprocess یک مؤلفه‌ی درخواست‌کننده است (با توجه به شکل (۱)) بنابراین طبق تعریف (۲) کنش s یک کنش خروجی است. امّا مؤلفه Mshift مؤلفه فراهم کننده‌است، بنابراین طبق تعریف ۱ کنش s برای مؤلفه‌ی Mshift کنش ورودی است. مؤلفه Mprocess کنش ورودی ندارد و مؤلفه‌آتاماتا Mshift کنش خروجی ندارد و هر دوی مؤلفه‌ها کنش داخلی ندارند. در ایجاد آتاماتای تیمی مؤلفه-آتاماتای Mprocess را با اندیس ۱و مؤلفه‌آتاماتای Mshift را با اندیس ۲ در نظر می‌گیریم.

کنش s یک کنش خارجی برای مؤلفه Mshift و Mprocess محسوب می‌شود بنابراین برای ایجاد آتاماتای تیمی SA-TA تعاملات روی این کنش باید به طریقی انجام شود که کنش s برای Mprocess نقش کنش خروجی و برای Mshift نقش کنش ورودی را ایفا کند.

آتاماتای SA-TA حاصل از سناریوی شماره ۱ در شکل (۵) نشان داده‌شده‌است در این شکل وضعیت‌ها حاصلضرب دکارتی وضعیت‌ها در مؤلفه‌آتاماتا Mshift و Mprocess هستند ترتیب آنها هم بر اساس اندیس‌هایی است که برای Mprocess و Mshift در نظر گرفته‌ایم مثلاً در وضعیت(S2,S3) S2 وضعیت مؤلفه‌آتاماتای Mprocess و S3 وضعیت مؤلفه‌آتاماتا Mshift است. در شکل (۵) در گذار

{(S1,S1) s (S2,S3)} }، گذار {S1, s ,S2} مربوط به مؤلفه‌آتاماتا Mprocess است که کنش s ، نقش کنش خروجی را دارد و گذار {S1, s ,S3} مربوط به مؤلفه‌Mshift را دارد که کنش s ، کنش ورودی است.

همان‌طور که بیان شد، در آتاماتای تیمی، کنش‌های داخلی هر مؤلفه آتاماتا منحصر به فرد هستند، بنابراین در همزمانی کنش‌های داخلی فقط یکی از مؤلفه‌ها شرکت می‌کنند و مؤلفه‌های دیگر بیکار خواهندبود. در ایجاد آتاماتای تیمی مؤلفهٔ بیکار را x در نظر می‌گیریم و از این طریق وضعیت‌های اضافی را حذف و به نوعی با یک وضعیت معادل آنها را نشان می‌دهیم. در شکل (۵) کنش p در مؤلفه‌آتاماتا Mprocess یک کنش داخلی است بنابراین فقط مؤلفه‌آتاماتا Mprocess در همزمانی این کنش با سایر مؤلفه‌ها شرکت می‌کنند.



شکل (۵): آتاماتای تیمی حاصل از سناریوی شماره ۱

نمونه‌های آزمون انتزاعی همان مسیرهای موجود در SA-TA است. که ما با در نظر گرفتن معیار شمول گذار[۲۱] نمونه‌های آزمون انتزاعی زیر را بدست آورده‌ایم.

- **(1)**: (S0, X), p, (S1, S1), s, (S2,S3) , c_h, (S4, X), h, (S5, X)
- **(2)**: (S0, X), p, (S1, S1), s, (S2, S3), u_h, (S3, X), r_h, (S2, S3)
- **(3)**: (X,S0), a_s, (S1, S1), s, (S2, S3), u_h, (S3, X), r_h, (S2, S3)
- **(4)**: (X, S0), a_s, (S1, S1), s, (S2, S3), c_h, (S4, X), h, (S5, X)
- **(5)**: (S0, X), p, (S1, S2), s, (S2, S3), u_h, (S3, X), r_h, (S2, S3)
- **(6)**: (S0, X), p, (S1, S2), s, (S2,S3) , c_h, (S4, X), h, (S5, X)
- **(7)**: (X,S0), u_s, (S1, S2), s, (S2, S3), u_h, (S3, X), r_h, (S2, S3)
- **(8)**: (X, S0), u_s, (S1, S2), s, (S2, S3), c_h, (S4, X), h, (S5, X)

برای ایجاد نمونه‌های آزمون واقعی با توجه به پیاده‌سازی سیستم منابع انسانی امین نگاشتی بین رویدادهای معماری(کنش‌ها) و توابع و ساختارهای پیاده‌سازی برقرار شده است و برای نمونه آزمون انتزاعی شماره۱ نمونه‌ی آزمون زیر بدست آمده‌است.

IndexAction, GetpersonenlInfo, Getshift,
Calcattend, Saveatten

۴- آزمون مرکب

در سیستم‌های مبتنی بر مؤلفه، گاهی بعضی از مؤلفه ها از چندین مؤلفه‌ی درونی مرتبط با یکدیگر، تشکیل شده است. در ادبیات آزمون نرم‌افزار، چنین مؤلفه هایی، مؤلفه های مرکب و آزمون چنین سیستم-هایی آزمون مرکب[۲۲] نامیده می‌شود.

در سیستم‌های مرکب ارتباطات گسترده و پیچیده‌ای وجود دارد؛ از یک طرف، ارتباطات بین مؤلفه‌ها (ارتباطات اسمبلی) و از طرف دیگر، ارتباطات هر مؤلفه‌ی مرکب با زیرمؤلفه هایش (ارتباطات وکالتی)، پیچیدگی تعاملات را بیشتر کرده‌است. برای ایجاد آتاماتای تیمی، در سیستم‌های مرکب الگوریتمی ارائه خواهد شد، که برای کاهش پیچیدگی، مؤلفه‌ها در سطوح مختلف سازمان دهی و بصورت سلسله مراتبی رسمی می‌شوند. این الگوریتم را "*رسمی‌سازی ترکیبی*" نامیده-ایم.

هر آتاماتای تیمی را می توان به عنوان یک مؤلفهٔ آتاماتا در نظر گرفت [9]. از این خاصیت در الگوریتم *رسمی‌سازی ترکیبی* برای ایجاد آتاماتای تیمی استفاده شده‌است. همچنین با توجه به اینکه در آتاماتای SA-TA کنش‌های ارتباطی غیرقابل مشاهده برای مؤلفه های دیگر تعریف شده‌است، از هر SA-TA به عنوان یک مؤلفه‌ی آتاماتا در سطوح بالاتر استفاده می‌شود. بنابراین، هر مؤلفه‌ی مرکب به عنوان یک مؤلفه‌ی آتاماتا با مؤلفه های دیگر ارتباط برقرار می‌کند.

برای نمایش ارتباطات بین مؤلفه‌های مرکب و زیر مؤلفه از گراف مؤلفه استفاده می‌شود. گراف مؤلفه، معماری سیستم را از جنبه‌ی ترکیب‌پذیری نمایش می‌دهد، در این گراف گره‌ها، مؤلفه‌های سیستم و لبه‌ها ارتباطات بین مؤلفه‌های مرکب و زیر مؤلفه‌ها را نشان می‌دهد. برای ایجاد گراف مؤلفه، از نمودار مؤلفه UML2.0 استفاده می شود. گراف مؤلفه بصورت زیر تعریف می شود.

$$Component - Graph = (UcomponentsCD, Rcomposite)$$

(۷) گراف مؤلفه

در گراف مؤلفه UcomponentsCD مجموعه‌ی متناهی از مؤلفه‌ها در نمودار مؤلفه UML2.0 است و Rcomposite مجموعهٔ متناهی از لبه‌هاست که ارتباطات بین مؤلفه های مرکب و زیر مؤلفه ها را نشان می‌دهد(ارتباطات وکالتی).

در الگوریتم پیشنهادی مؤلفه‌ها در گراف مؤلفه بصورت عمق اول پیمایش می‌شوند. هر مؤلفه به صورت مؤلفه‌آتاماتا مدل می‌شود و مؤلفه‌های آتاماتا با SA-TA تشکیل آتاماتای تیمی می‌دهند. در صورتی که مؤلفه مرکب باشد. آتاماتای تیمیSA-TA با زیر مؤلفه‌های آن تشکیل می‌شود و آتاماتای تیمی حاصل از این زیرمؤلفه‌ها با مؤلفه-های آتاماتا در سطوح بالاتر تعامل برقرار می‌کند. این روند بصورت بازگشتی ادامه می‌یابد تا توصیفات معماری نرم‌افزار رسمی شود. سپس نمونه‌های آزمون انتزاعی از توصیفات رسمی استخراج می‌شود. در نهایت هم نمونه‌های آزمون انتزاعی به نمونه‌های آزمون واقعی تبدیل

می‌شوند. شبه کد الگوریتم رسمی‌سازی ترکیبی را در قسمت ضمایم آورده‌ایم.

۵- نتیجه‌گیری و کارهای آینده

روش پیشنهادی یک راهکار زیر بنایی است که زمینه را برای آزمون خودکار نرم‌افزار فراهم می‌کند. در روش پیشنهادی توصیفات ساختاری و رفتاری سیستم توسط معماری نرم‌افزار ارائه می‌شود. با استفاده از معماری نرم‌افزار، آزمون نرم‌افزار می‌تواند همزمان با فرآیند ساخت نرم-افزار ایجاد شود بنابراین خطاها زودتر کشف می‌شوند و اصلاح آنها ارزان‌تر خواهدبود.

به دلیل اینکه، مدل‌های رسمی امکان خودکارسازی آزمون نرم‌افزار فراهم می‌کنند، بنابراین در این مقاله بیشتر روی رسمی سازی معماری نرم‌افزار متمرکز شدیم و از آتاماتای تیمی به سبب ویژگی‌های منحصر به فردش برای این منظور استفاده‌کرده‌ایم. آتاماتای تیمی یک مدل رسمی است. بنابراین ابزارهای کارآمدی برای ایجاد آن می‌توان تولید کرد که مکمل روش پیشنهادی باشد. همچنین مبنای آتاماتای تیمی ماشین حالت محدود است ماشین حالت محدود است مبنای بسیاری از زبان‌های توصیف معماری هستند. بنابراین، روش پیشنهادی برای اکثر زبان توصیف معماری کاربرد دارد.

روش پیشنهادی یک روش عمومی و کاربردی است که در حوزه‌های دیگر نیز می‌تواند مورد استفاده قرار گیرد. روش پیشنهادی در حوزه‌ی تصدیق و تایید نیز می‌تواند مورد استفاده قرارداد.

در روش پیشنهادی، کنش‌ها را ترتیبی و پشت سر هم در نظر گرفته‌ایم، بنابراین روش پیشنهادی برای سیستم‌های همروند و موازی مناسب نیست. در کارهای آینده، آزمون نرم‌افزار این سیستم‌ها بیشتر مورد مطالعه قرار خواهد گرفت. نمونه‌های آزمون انتزاعی نیز می‌تواند ورودی به یک سیستم مدل رانده باشد و از این طریق نمونه‌های آزمون واقعی، خودکار ایجاد شوند.

ضمایم

الگوریتم ۱: رسمی‌سازی ترکیبی

```
Input: Main component
Output: SA-TA of subcomponents of Main component
Function CompositeFormalism (component )
begin
        SetChild={ the set of child nodes of Main component }/* this set is calculated in the first step */
        while (SetChild is not empty) do
                Let N a node in SetChild, and corresponding to the component C ;
                if (the SA-TA associated to N exists, or N is a leaf node )
                 then
                        SettoFormal = SettoFormal ∪ {C}
                        SetChild = SetChild \{N}
                else
                        CA= CompositeFormalism (C) /* calculate the SA-TA corresponding to N*/
                endif
        end while
        CA = formalism( C1,C2,.....), where Settoformal = { C1,C2,.....}/* call to the function formalism in order to
        formalism the components in the set SettoFormal*/
        return SA-TA;
end
```

مراجع

[۱] شرفی، سید مهران، **ارائه روشی جهت استخراج و ارزیابی ویژگی-های غیروظیفه‌مندی مبتنی بر توصیفات رسمی معماری نرم-افزار**، دکتری، دانشگاه آزاد اسلامی واحد علوم و تحقیقات، تهران، ۸۶.

[2] Bertolino A., *Knowledge Area Description of Software Testing*, SWEBOK, Joint IEEE-ACM Software Eng. Coordinating Committee, http://www.swebok.org, 2000.

[3] Garlan D., Shaw M., *Software architecture: perspective on an emerging discipline*. Prentice Hall, 1996.

[4] Legeard B.,Utting M., *Practical Model-Based Testing* Morgan Kaufmann Publishers, 2006.

[5] Pressman, R S., *Software Engineering - A Practitioner's Approach*, McGraw Hill, 2004

[6] Muccini H., Dias M., Richardson D.J., *"Software architecture-based regression testing"*, Journal of Systems and Software, vol. 79, No. 10, pp. 1379-1396, October 2006.

[7] Grant E.S., Reza H., *"A method to test concurrent systems using architectural specification"*, The Journal of Supercomputing, vol. 39, No. 3, pp. 347-35, 2007.

[8] Muccini H., Bertolino A., Inverardi P., "*Using software architecture for code testing"* , IEEE Transactions on Software Engineering, vol. 30, No. 3, pp. 160 - 171 , March 2004.

[9] terBeek M.H., Ellis C.A., Kleijn J., Rozenberg G., "*Synchronizations in Team Automata for Groupware"*,

Computer Supported Cooperative Work—The Journal of Collaborative Computing, vol. 12, No. 1, pp. 21-69, 2003.

[10] Zhenyi J., Offutt J., "*Deriving Tests From Software Architectures*", Proceedings in 12th International Symposium on Software Reliability Engineering, 2001.

[11] Reza H., Lande S., *"Model Based Testing Using Software Architecture"*, proceeding in Information Technology: New Generations (ITNG), pp. 188 – 193, 2010.

[12] Garlan D., Monroe R.T., Wile D., "*ACME: An Architecture Description Interchange Language*". Proceedings of CASCON 97, pp. 169-183, 1997.

[13] Sharafi M., Shams Aliee F., Movaghar A., *"A Review on Specifying Software Architectures Using Extended Automata-Based Models"*, in FSEN 2007, pp. 423–431, 2007.

[14] Ivers J., Clements P., Garlan D., Nord R., Schmerl B., viedo Silva J. R., *Documenting Component and connector Views with UML2.0*, Technical report, CMU/SEI, TR-008 ESC-TR-2004-008, 2004.

[15] Object Management Group. *Unified Modeling Language: Superstructure*, Version 2.0, ptc/03-07-06, July 2003, http://www.omg.org/cgi-bin/doc?ptc/2003-08-02.

زیرنویس‌ها

[1] Software Test
[2] Test Cases
[3] Software Architecture
[4] Labeled Transition System
[5] Model-Based-Testing(MBT)
[6] United Modeling Language
[7] Team automata
[8] Abstract test case
[9] Sequence Diagram
[10] Component Diagram
[11] State machine
[12] Assembly Connector
[13] Delegation Connector
[14] Component provider
[15] Component requestor
[16] Component automata
[17] در این مرحله ارتباطات اسمبلی مدل می‌شوند و در الگوریتم رسمی-سازی ترکیبی ارتباطات وکالتی مدل می‌شوند.
[18] complete transition space
[19] coverage criteria
[20] Composite Component
[21] Transition Coverage
[22] Compositional Testing